

# Can Co-robots Learn to Teach?

Harshal Maske, Emily Kieson, Girish Chowdhary, and Charles Abramson

**Abstract**— We explore beyond existing work on learning from demonstration by asking the question: “Can robots learn to teach?”, that is, can a robot autonomously learn an instructional policy from expert demonstration and use it to instruct or collaborate with humans in executing complex tasks in uncertain environments? In this paper we pursue a solution to this problem by leveraging the idea that humans often implicitly decompose a higher level task into several subgoals whose execution brings the task closer to completion. We propose Dirichlet process based non-parametric Inverse Reinforcement Learning (DPMIRL) approach for reward based unsupervised clustering of task space into subgoals. This approach is shown to capture the latent subgoals that a human teacher would have utilized to train a novice. The notion of “action primitive” is introduced as the means to communicate instruction policy to humans in the least complicated manner, and as a computationally efficient tool to segment demonstration data. We evaluate our approach through experiments on hydraulic actuated scaled model of an excavator and evaluate and compare different teaching strategies utilized by the robot.

## I. INTRODUCTION

In many real world robotic applications, human operators play a critical role in ensuring the safety and efficiency of the task. Some examples include heavy construction and agricultural robotics where human operators of co-robots such as excavators, tractors, and backhoes must make safety-critical decisions in real-time under uncertain and dynamically changing environments. The *skill-gap* between expert and novice operators of these robots is a significant limiting factor in ensuring safety, efficiency, and quality at work-sites. If a co-robot was able to learn from experts and utilize that knowledge to assist or teach novice operators, significant performance gains could be achieved. In this paper, we study the crucial problem of directly learning instruction policies for novice operators from demonstrations provided by skilled operators. Learning from Demonstration has been widely studied in the context of robots learning to do a task from teacher demonstrations [2]. However, when a robot needs to teach a human operator, the robot needs to do much more than just learning to imitate the demonstrated task. Rather, it has to simplify and decompose the tasks into human understandable task-primitives, and communicate back the essential sequence of actions to guide the human learning from the robot. This brings us to the very crucial question: How can Robots learn to Teach? We argue that there are

Harshal Maske, and Asst. Prof. Girish Chowdhary\* are with the Distributed Autonomous Systems laboratory (DASLAB), University of Illinois at Urbana Champaign, {hmaske2@illinois.edu, girishc@illinois.edu}. Emily Kieson, and Prof. Charles Abramson are with the Comparative Psychology Laboratory, Oklahoma State University, {kieson@okstate.edu, charles.abramson@okstate.edu}.

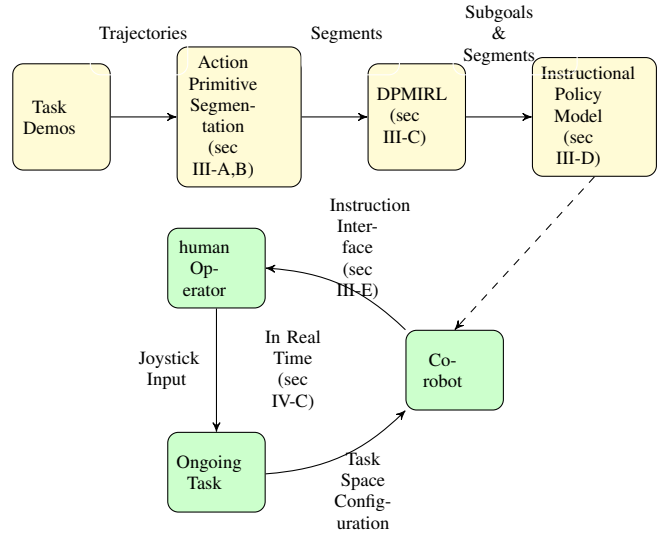


Fig. 1: Co-robot learns instructional policy from expert demonstrations (off-line learning in yellow). In real time (shown in green), co-robot generates instruction/guidance for human operators based on the current task space configuration and the instructional policy model.

two important aspects to answering this question: First is the development of practical algorithms that would allow a co-robot to extract latent subgoals or skills that a human teacher would have utilized to instruct other humans. Second, is the development of feedback strategies for providing the appropriate task specific guidance to the human based on the current task state. The approach formulated in this paper is designed to address both these aspects and is shown through extensive experimentations to enable robots to teach complex tasks to human operators (see figure 1).

The main contribution of this paper is a method to directly learn an instructional policy from human demonstrations. We define an instructional policy as a feedback policy that utilizes the robots current and past state to suggest the best set of future actions in order to proceed with a given task. This should be contrasted with existing LfD work that has focused primarily on the robot learning a policy for executing the task by itself. Yet, our approach is highly scalable and generalizable, and has been demonstrated to work on a realistic LfD problem with multiple degrees of freedom and uncertain operating conditions. Hence, it can also be used in a pure LfD form for complex real-world robotic tasks, such as those often encountered in construction. To ensure scalability and generalizability as well as to simplify

communication with the human learner, we introduce the notion of *action primitives* which are used in unsupervised segmentation of demonstration trajectories. Unlike the motion primitives, action primitives are defined in the joint space of a robot, and are hence universally generalizable to any articulated robot, allowing explicit segmentation of demonstration data without the need for computationally expensive and time consuming MCMC sampling that has been utilized by existing LfD approaches.

The developed approach is demonstrated in a rigorously designed large human-robot experiment with 113 human participants learning to perform the truck loading task with a scaled excavator in a different configuration (each time). Demonstrations from experts were collected and utilized by our algorithms to learn the instructional policy. Two different interfaces of communicating the correct action primitives to the human operators are investigated, and shown to lead to statistically significant improvement in the human learner's task completion time, task safety, and retention. Although all of the experiments are done with a scaled excavator, the results should translate well to working with real excavators, and have immediate significance for teleoperated robotics. In fact, we note that working with a scaled excavator can be more cognitively demanding than working from within a real excavator because the operator has to transpose themselves to the correct reference frame as the turret rotates.

## II. BACKGROUND

### A. Learning from Demonstration

Much of the existing work in LfD has focused on the problem of enabling the robot to learn a closed loop policy to perform a set of actions on its own. LfD has been successful in teaching robots tennis swings [20], walking gaits [12], and complex helicopter maneuvers [1]. A review of various LfD techniques is available in [2]. In a subset of methods called Inverse Reinforcement Learning (IRL), the reward function is learned from a set of expert demonstrations [1]. Learning reward is argued by [11] to be equivalent to high-level description of the task, that explains the expert's behavior in a richer sense than the policy alone.

Some recent work in LfD has focused on performing automatic segmentation of demonstrations into simpler reusable primitives [8], [5], [3], [15]. The work by [14] uses the Beta Process Autoregressive Hidden Markov Model (BP-AR-HMM) developed by [4] to perform auto-segmentation of time series data available from multiple demonstrations. For sequencing [3], [5] developed novel non-parametric probabilistic models in contrast [14] developed a finite state automaton that utilizes pose information of task objects and segment lengths for sequencing. In these methods, the segments or task primitives are loosely defined, with no bounds on the number of possible segments, thus limiting their re-usability. In this paper, we first propose a definition for segments, that is then utilized to perform segmentation of an unstructured demonstration. Moreover, this procedure does not rely on computationally inefficient Gibbs sampling for learning model parameters. Aforementioned algorithms

attempt to directly learn the policy; next we discuss reward based IRL techniques for LfD. Our attempt is to leverage key concepts from these two school of thoughts to develop algorithm for inverse LfD to enable robots to assist or instruct humans.

### B. Bayesian Nonparametric Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL) is an LfD technique concerned with finding hidden reward function of an expert human demonstrator from the demonstrated state and action samples [22], [1]. Recently, an approach to solve IRL by automatically decomposing the reward function into a series of subgoals, which was viewed as local reward functions, was proposed [11]. We utilize the notion of executing subgoals in a particular sequence to perform a task as a key component of instruction policy model. This is based on research that deals with human expertise in complex environment [16], [6]. According to these findings, humans often form implicit decompositions of higher level tasks into several subgoals, so that the execution of each subgoal brings the task closer to completion.

In IRL [1] a Markov Decision Process (MDP) without the reward function  $R(s)$  i.e.  $MDP \setminus R$  is given. A demonstration set  $O$  consists of state action pairs,  $O = \{(s_1, a_1), \dots, (s_N, a_N)\}$ , where each pair  $O_i = (s_i, a_i)$  indicates that the action  $a_i$  was performed from the state  $s_i$ . Multiple set of demonstrations are used as an input to the IRL algorithm to obtain an estimate of reward function  $\hat{R}(s)$ , such that the corresponding optimal policy  $\pi^*$  matches the observations. The IRL problem is ill-posed, since defining reward as,  $\hat{R}(s) = c \forall s \in S$ , would make any set of state-action pairs trivially optimal. Moreover, it is possible to encounter dissimilar actions from a particular state  $s_i$ . This ambiguity was resolved by restricting the reward function to be of certain form [19], [21], [13]. Later [18] developed a standard Bayesian inference procedure to learn reward function.

The IRL methods cited above attempt to explain the entire observations set  $O$  with a single, complex reward function resulting in a large computational burden when the space of candidate reward functions is large. To overcome this limitation, Michini et. al. [11] developed Bayesian non-parametric IRL (BNIRL) that partitions the demonstration set  $O$  and explains each partition with a simple reward function or a "subgoal"  $R_g(s)$ , which consists of a positive reward at a single coordinate  $g$  in the state (or feature) space (zero elsewhere). A Dirichlet process prior (Chinese Restaurant process (CRP) construction) is assumed over the unknown number of partitions that consists of observed state-action pairs  $O_i \equiv (s_i, a_i)$ . Partition assignment for each observation  $O_i$  is denoted by  $z_i$ . Posterior over the partition assignment is given by

$$P(z_i | z_{-i}, O) \propto \underbrace{P(z_i | z_{-i})}_{\text{CRP}} \underbrace{P(O_i | R_{z_i})}_{\text{likelihood}} \quad (1)$$

where the first term denotes standard CRP prior and the second term evaluates the likelihood of the action  $a_i$  given the

subgoal reward function  $R_{z_i}$  corresponding to partition (or subgoal) identified by  $z_i$ . This likelihood term is evaluated using exponential rationality model (similar to that in [18]):

$$P(O_i|R_{z_i}) = P(a_i|s_i, z_i) \propto e^{\alpha Q^*(s_i, a_i, R_{z_i})} \quad (2)$$

where the parameter  $\alpha$  represents the degree of confidence in the demonstrator's ability to maximize reward. The evaluation of optimal action value function  $Q^*$  requires substantial computation and becomes infeasible for large state spaces. Hence, the author developed an approximation based on action comparison for the action likelihood (2), as follows

$$P(O_i|R_{z_i}) = P(a_i|s_i, z_i) \propto e^{\alpha \|a_i - a_{CL}\|_2} \quad (3)$$

where  $a_{CL}$  is the action given by some closed-loop controller attempting to go from state  $s_i$  to subgoal  $g_{z_i}$ . Note that  $z_i$  is a partition assignment variable for state  $s_i$ , if  $z_i = k$ , then  $g_k$  is the coordinate of  $k^{th}$  subgoal. This approximation to BNIRL [10], enables successful application of IRL to real-world learning scenario characterized by large state space such as quad-rotor domain. However, their approach was tested for a very basic maneuver of traversing four way-points in space. We utilize the structure of BNIRL, particularly the partitioning of high dimensional state space into finite subgoals, and propose a simplification that extends its application to more complex tasks in real world settings.

### C. Dirichlet Process Means for Gaussian mixture model

Dirichlet process (DP) is a well known prior for the parameters of a Gaussian mixture model when the number of mixture components are not known a-priori. Recently Kulis and Jordan [9] have shown that the Gibbs sampling algorithm for the Dirichlet process mixture approaches a hard clustering algorithm when the covariances of the Gaussian variables tend to zero. This results in a k-means-like clustering objective given by

$$\min \sum_{c=1}^k \sum_{x \in l_c} \|x - \mu_c\|^2 + \lambda k$$

where  $\mu_c = \frac{1}{|l_c|} \sum_{x \in l_c} x$

where  $k$  is the number of clusters,  $\mu_c$  is the mean for cluster  $c$ , and  $l_c$  is the set of data points  $x$  that belongs to the cluster  $c$ . This objective function is similar to that of k-means but includes a penalty  $\lambda$  for the number of clusters. DP-means algorithm behaves similarly to that of k-means with the exception that a new cluster is formed whenever a point is farther than  $\lambda$  away from every existing cluster centroid. The algorithm is initialized with a single cluster whose mean is simply the global centroid. We use this approach to cluster the states by assuming Gaussian distribution over their euclidean norm.

## III. METHODOLOGY

In LfD literature dynamic movement primitive (DMP) framework [7] is a prevalent approach to imitate demonstrations. Given the start and goal position in task space, a robot uses the DMP to generate trajectory (or set of states  $s$ ) for end-effector position and its mechanism, which is

then mapped to appropriate joint angle (or set of actions  $a$ ) using inverse kinematics [17] and thus the entire state-action policy map. We are interested to generate instruction policy for operators who perform the task, for which DMPs, which are encoded in terms of the trajectory, may not be the best solution. To track a given trajectory (i.e. the set of states  $s$ ) in 3-D space is a complex task for an operator and therefore can present difficulties in teaching. On the other hand, instructions in terms of joystick movements are easily understandable by the operator and are arguably the simplest means of communicating desired actions to operators. So why can't we use the joint angles i.e. the set of actions  $a$  from DMP? We could, since the joint angles can be mapped to joystick movement, however in order to do so, we need to perform two complex transformations ( $s$  to  $a$  and  $a$  to joystick actuations). Furthermore, it is not desirable to continuously instruct joystick movement to reach the goal from a start position. To tackle this issue, we introduce the notion of action primitives designed to simplify robot to human teaching tasks.

### A. Action Primitives

The definition of action primitives is developed for articulated robots, however, it should be generalizable to other types of robots. For an articulated robot, a revolute joint provides single-axis rotation function, hence a joint  $j$  can take three broadly defined possible states: i) counterclockwise rotation, ii) stationary or non-zero noisy perturbation, or iii) clockwise rotation. Note that it is possible to add further resolution to this decomposition. Let  $r_j$  be the variable that takes on values from the set  $\{1, 2, 3\}$  corresponding to these states respectively. Thus  $r_j = 1$  implies that the joint  $j$  is rotating counterclockwise and so on. The assignment of values  $\{1, 2, 3\}$  is arbitrary. *Mathematically, an action primitive defines the action of a robot in terms of variable  $r_j$  for each joint  $j$ .* Action primitives are defined in the joint space of a robot and hence map directly to joystick movements. Secondly, in this formulation, an action primitive changes only when the direction of rotation, i.e. the variable  $r_j$  changes for a joint  $j$ . This has two significant consequences: First an action primitive can be used to decompose a demonstration consisting of continuous state action spaces into finitely many discrete state-action pairs, this is the basis for segmentation approach discussed in next section. Second, known sequence of these finite number of action primitives can be used to generate step-wise instruction policy model to guide humans (discussed further in section III-D). An example of action primitive  $a_\tau$  at a time  $\tau$  for a robot with  $n = 3$  revolute joints is  $a_\tau = [r_1 = 1; r_2 = 2; r_3 = 2]$  or just  $a_\tau = [1, 2, 2]^T$  indicative of counterclockwise rotation for the first revolute joint. Clearly for a robot with  $n$  joints, there would exist finitely many action primitives, as opposed to infinitely many joint velocities obtained from DMPs.

### B. Action Primitive based Segmentation of Task demos

Action primitives are closely related to the joint space of a robot and is a least complicated mechanism to commu-

nicate instructions to humans when compared to dynamic motion primitives or any other existing methods because they relate directly to joystick motion. We now describe how action primitives can be used for segmenting demonstrations. Demonstration data in the form of joint positions  $x_\tau$ , and joint velocities  $v_\tau$  sampled from a continuous demonstration of a task at time instants  $\tau = 1, 2, \dots, T$ , is used as an input to the segmentation algorithm. Classification of sampled velocities into action primitives generates the required segmentation. Each segment will be an action primitive that spans over finite instants of time. But first we need to define distribution for each action primitive class. Recall that an action primitive is defined by the value of its variable  $r_j$  for each joint  $j$ . As noted earlier each  $r_j \in \{1, 2, 3\}$ , hence we need three distributions for each joint  $j$  of the robot. To accomplish this, we cluster sampled velocities assuming Gaussian distribution for each cluster.

Let  $v_\tau \in \mathbb{R}^{n \times 1}$  denote the sampled velocity at time  $\tau$ ,  $v_{\tau j}$  be the velocity of joint  $j$ , and  $v_j$  be the set of all sampled velocities  $\{v_{1j}, v_{2j}, \dots, v_{Tj}\}$  of the joint  $j$ . We cluster velocities in the set  $v_j$  using k-means algorithm by setting  $k = 3$  to obtain three clusters corresponding to  $r_j \in \{1, 2, 3\}$ . Our results show that applying k-means consistently produces clusters such as one shown in figure 2, which clearly demarcates the intended three different states for  $r_j$ . From cluster members we calculate the mean  $\mu_{ji}$  and variance  $\sigma_{ji}^2$  for each cluster  $i \in \{1, 2, 3\}$ . Based on these cluster parameters we define the probability

$$p(r_j = i | v_{\tau j}) = \mathcal{N}(v_{\tau j} | \mu_{ji}, \sigma_{ji}^2) \quad (4)$$

and assign  $r_j$  as follows

$$r_j = \begin{cases} 2 & p(r_j = 2) > \eta \\ 1 & p(r_j = 1) > p(r_j = 3) \\ 3 & p(r_j = 3) < p(r_j = 1) \end{cases} \quad (5)$$

where  $\eta$  is a threshold which can be set using a labeled trajectory data. This procedure is repeated for each joint  $j$  to obtain the action primitive  $a_\tau$  at each time instant  $\tau$ . This process generates action primitive segments  $\mathcal{A}_i$  comprised of similar action primitive  $a_\tau = a_i$  observed continuously over time instants  $\tau = \{\tau_i, \tau_{i+1}, \dots, \tau_j\}$ . An example segmentation (discussed later in details) is shown in figure 6, where each colored segment is an action primitive segment ( $\mathcal{A}_i$ ) that has a particular action primitive ( $a_i$ ). Total of 14 unique action primitives (or action primitive segments) were discovered in a truck loading task performed using a model excavator having four revolute joints. Repetition of these unique action primitive segments generates the entire task. We associate each action primitive segment  $\mathcal{A}_i$  with the corresponding end-effector pose  $s_i$  at the beginning of an action primitive. Each action primitive segment can then be represented by the pair  $(s_i, a_i)$  where  $a_i$  is the associated action primitive. Thus using segmentation procedure we obtain the set state-action pairs  $O = \{(s_1, a_1), (s_2, a_2), \dots, (s_N, a_N)\}$ , where  $N$  is the total number of action primitive segments observed in a demonstration. The set  $O$  is used as an input to the

inverse reinforcement learning described in the next section.

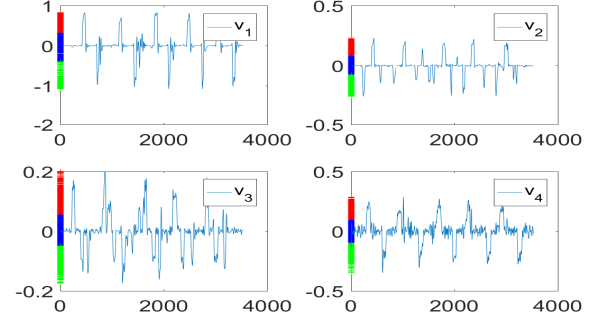


Fig. 2: k-means clustering of sampled velocities for each of the four joints of model Excavator with  $k = 3$ . Clustering demarcates the intended three different states for Learning (DPMIRL)

We had noted earlier that humans are good at decomposing a given ill-defined task into a series of actionable subgoals. Such decomposition is implicit within the human mind, and humans are not always able to clearly explain how they arrived at the decomposition [16], [6]. However, the key idea that is leveraged is to decompose complex task space into subgoals. This is achieved through DPMIRL formulation: a reward based partitioning of task space. Additionally action primitive segments are associated with the inferred subgoals, such that their proper sequencing generates instructions to translate from one subgoal to another.

We first propose a simplification of action likelihood approximation used in the BNIRL approach (discussed in section II-B). This simplification allows partitioning of state-action pairs based on euclidean distance metric using Dirichlet process means [9]. Typically in an IRL problem entire state space is modeled as an MDP, whereas we define an  $MDP \setminus R$  as a tuple  $\langle S, A, T, \gamma, D \rangle$  where the sets  $S$  and  $A$  consists of the state action pairs in the set  $O$  which is obtained from action primitive based segmentation. This step significantly reduces the computational burden, and potentially renders the analysis of any problem in high dimensional continuous state action space feasible. We simplify the likelihood term in equation (3), where the likelihood of an action  $a_i$  w.r.t to a subgoal  $g_{z_i}$  is computed as  $P(a_i | s_i, z_i) \propto e^{\alpha \|a_i - a_{CL}\|^2}$ . As proposed in [10],  $a_{CL}$  is the action of a closed-loop controller that attempts to go from  $s_i$  to subgoal  $g_{z_i}$ . A simple controller that would generate an action  $a_{CL}$  to reduce the pose error between the subgoal  $g_{z_i}$  and the present state  $s_i$  is  $a_{CL} \propto (g_{z_i} - s_i)$ . Since a demonstrator is invariably reducing the pose error between his/her state  $s_i$  and the subgoals, we argue that the action  $a_i$  in the demonstration is also proportional to the pose error. Hence given a particular subgoal  $g_{z_i}$ , even  $a_i \propto (g_{z_i} - s_i)$ . Substituting in equation (3) we obtain

$$P(a_i | s_i, z_i) \propto e^{\alpha \|\kappa(g_{z_i} - s_i) - \lambda(g_{z_i} - s_i)\|^2} \quad (6)$$

$$\propto e^{\alpha' \|(g_{z_i} - s_i)\|^2} \quad (7)$$

where we let  $\kappa$  and  $\lambda$  to be scalar proportionality constants, and thus the original action comparison reduces to pose error

comparison between the current state  $s_i$  and the subgoals. This result is very intuitive in the sense that any state-action pair  $(s_i, a_i)$  will be partitioned or assigned to the closest subgoal. Thus any action  $a_i$  from a state  $s_i$ , is likely to either attain the assigned subgoal or recede away towards the next. This notion is utilized to partition the state action pairs in the set  $O$ , using euclidean distance metric on state locations i.e.  $\|s_i\|_2$  for a state  $s_i$ . This is performed using DP-means algorithm discussed in section II-C. Thus we obtain clusters  $\{c_1, \dots, c_k\}$  where the number of clusters  $k$  is not known a-priori, and each cluster  $c_j$  consists of member states  $\{s_i \in c_j : z_i = j\}$ . Since each state  $s_i \in c_j$  is associated with an action primitive  $a_i$ , we define a set  $S_j$  w.r.t the cluster  $c_j$  as  $\{(s_i, a_i) \in S_j : s_i \in c_j\}$ . We define the subgoal as a multivariate Gaussian  $X_j \sim \mathcal{N}(\mu_j, \Sigma_j)$  in  $n$ -dimensional space (formed by end-effector position and mechanism) whose parameters are obtained from the member states of cluster  $c_j$ .

To ensure that the instruction policy for a given task, is generalizable to a novel task configuration it is important to define appropriate reference coordinate frames to compute the euclidean norm. Given a task configuration we define a coordinate frame centered on each known object. Let  $M$  be the number of objects and  $m_j$  be the center of the  $j^{th}$  object's coordinate frame w.r.t the base frame of the robot. We divide the states in the set  $S$  into  $M$  disjoint sets  $\{S_1, \dots, S_M\}$ , where each  $s_i$  is assigned to  $S_j$  such that  $\arg \min_j \|s_i - m_j\|_2$ . We run DP-means separately for each set  $S_j$ , to obtain subgoals as described in algorithm 1. Thus we have decomposed the task space into finite subgoals, most importantly defining subgoals as a multivariate Gaussian allows us to evaluate the likelihood of any state  $s$  in space w.r.t to each subgoal, to determine the most likely subgoal. This assignment allow us to select sequence of action primitives as instructions to reach next subgoal, this is discussed next.

---

#### Algorithm 1 DPMIRL

---

**Input:** Action primitive segments  $(S, A) \in O$ ,  $M$  objects of interest centered at  $m_j$ .

- Assign state  $s_i$  to  $S_j$  s.t.  $\arg \min_j \|s_i - m_j\|_2$
- Run DP-means algorithm for states in each  $S_j$ , to obtain set of clusters  $C_j = \{c_{j1}, \dots, c_{jk}\}$
- Set of subgoals  $X = \emptyset$ ,  $p = |X| = 0$

**for** each set of clusters  $C_j$  **do**

**for** each cluster  $k$  in  $C_j$  **do**

- Compute mean  $\mu_{kj}$  and variance  $\Sigma_{kj}$  for the member states  $s \in c_{jk}$
- Add  $X_p \sim \mathcal{N}(\mu_{kj}, \Sigma_{kj})$  to set  $X$ , increment  $p$ ,

**end for**

**end for**

**Output:** Set of subgoals  $X$ .

---

#### D. Task-come-Instruction Model

DPMIRL partitions the task space into subgoals this is analogous to expert human operators who also decompose

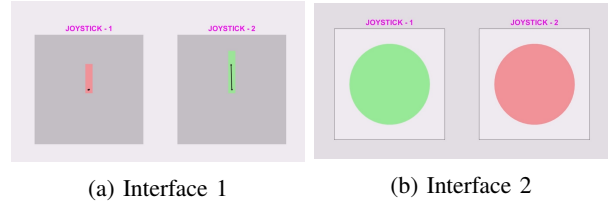


Fig. 4: Visual interface for instructing humans: a) The interface depicts desired joystick movement direction and magnitude using red colored bars with appropriate length. Thin black line corresponds to the actuation by the operator, which when matched, turns the red colored bar to green. b) Two circles correspond to each joystick. They turn green if the operator takes the desired action, and remain red otherwise. The goal is to reinforce desired behavior with minimal communication.

a given loosely defined task (such as leveling a construction site or loading a truck) into a series of actionable subgoals. This analogy is used as a building block for the instruction model that can be used by a robot to instruct or guide human operator. Action primitive based segmentation plays an important role in such a model, as the transition between these action primitives results in a transfer from one subgoal to another. Further an action primitives can be easily communicated to the human operator as they represent activation of single or multiple actuators which are in turn operated by specific joysticks. An example of action primitive and its corresponding communication in terms of joystick movements via a simple graphical interface is depicted in figure 4a.

We now elaborate the model construction which is generated autonomously based on the subgoals obtained using DPMIRL and the action primitive based segments. Construction of this model has to be such that the task is elaborated in the form of transition among the subgoals. One such construction using hierarchy of Markov chains also known as a Dynamical Bayesian Network is shown in figure 3, where the transition between subgoals is modeled by the topmost Markov chain. In this construction, we utilize a fact that each subgoal ( $X_i$ ) generated using DPMIRL has an associated set  $S_i$  of state-action primitive pairs, and there exists a Markov chain of variables  $\{Z_1^i, \dots, Z_{T_i}^i\}$ , where each variable is an action primitive at time instants  $\tau = 1, \dots, T_i$ , that results in a translation to the next subgoal  $X_j$ . Hence the Markov chain under each subgoal  $X_i$  models the transition among the action primitives associated with that subgoal. These transition models for action primitives are obtained from the segmentation of demonstration data and counting the transitions between the action primitive segments. The final layer of the model is actuator velocity variable  $y_\tau^i$  that is conditional on the action primitive  $Z_\tau^i$ , and is modeled as a Gaussian distribution over the sampled actuator velocities contained in the action primitive segment. Given the most likely subgoal ( $X_i$ ) for the current state  $s$  and the previous action primitive  $Z_{\tau-1}^i$ , the generative model for getting



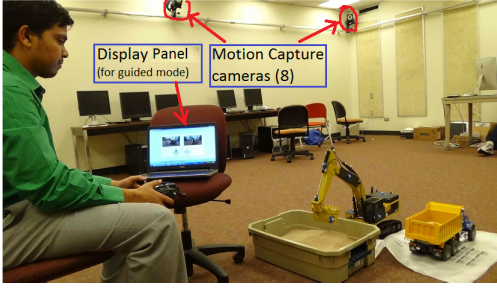


Fig. 5: Experimental set-up: Motion capture system, scaled excavator model, and display panel for guided demos.

instruction in the form of next action primitive  $Z_\tau^i$ , using the model in figure 3, is as follows:

$$P(X_{i+1}|X_i) \sim \Pi \quad (8)$$

$$P(Z_\tau^i|Z_{\tau-1}^i, X_i) \sim \pi(X_i) \quad (9)$$

$$P(y_\tau^i|Z_\tau^i) \sim F(\theta_{Z_\tau^i}) \quad (10)$$

and the parameters for this model are the transition distribution  $\Pi$  for subgoals, the subgoal specific transition model  $\pi(X_i)$  for the associated action primitives, and the parameter vector  $\theta_{Z_\tau^i}$  that models conditional distribution of actuator velocities given the action primitive.

#### E. Instruction Interface

Human robot interaction (HRI), an entire field of research is devoted to investigate most effective human-robot interfaces. A survey of the techniques typically utilized there is beyond the scope of this work. Rather, in this work we resort to exploratory but exhaustive evaluations using two fundamentally different styles of visual interfaces to communicate the instructional policy learned using methods developed in the previous sections. Our goal is to compare generic reward based interfaces against specific instructional interfaces. Given the nature of uncertainty and human presence around construction equipments, safety and situational awareness of the operator becomes another key aspect for the interface design. It is desirable therefore to have simple visual interface that reinforce desired skills while demanding minimal operator attention. On the other hand, the task is complex, so common wisdom is to err towards providing specific instructions on which actions to take. Accordingly, the first interface indicates the extent of desired actuation, current joystick position along with positive reinforcement by turning red colored bar to green, when an operator takes a desired action. This interface trades off operator's attention to communicate more information. On the other hand, the second interface provides positive reinforcement for desired actuation by changing the color of a circle representing each joystick. It does not provide any other information in terms of the direction on magnitude of joystick motion.

### IV. EXPERIMENTS

#### A. Test Platform

To test the developed approach exhaustive experiments were performed on a  $1/14^{th}$  scaled 345D Wedico exca-

TABLE I: Comparison of Action primitive based segmentation with BP-AR-HMM [14] in terms of computation time

No. of Time Series	2	4	6
Data size N 4-D data	6690	14033	23341
Action Primitive	21 sec	41 sec	68 sec
BP-AR-HMM	1639 sec	6690 sec	14752 sec

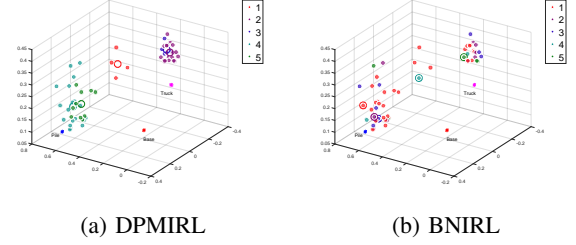


Fig. 7: Decomposition of task space into subgoals. Task space depicts robot's base, pile and truck location. Both algorithms discovered 5 subgoals shown as big circles in different colors. The action primitive segments associated with each subgoal are highly un-uniform for BNIRL an. Each subgoal is a mean location of its member states shown as filled circles with similar color.

vator model a 4 d.o.f robotic arm manipulator, controlled by a radio transmitter as seen in Figure 5, this model is constructed by Wedico, Germany (<http://www.wedico.de/>), which specializes in to-scale accurate and fully functional hydraulic construction equipments. Ideally the operator and the display panel shown in the figure 5 would be set up inside the excavator but this has been left for future work. However, the results reported here should generalize, furthermore, they are directly applicable to teleoperated robotics. The Wedico excavator lacked joint-angle encoders and internal proprioception, hence all the experiments were performed inside a motion capture facility to ensure real-time data input to the algorithm. In our experiments we record positions and velocities of the four actuators turret, boom, arm and bucket and denote them by vectors  $x$  and  $v$  respectively. We also record end-effector position w.r.t the base frame and bucket angle as four dimensional vector  $s^e$ .

#### B. Learning Instruction Policy

To demonstrate our approach we selected truck loading task which is a standard task performed using an excavator, and also happens to be a benchmark operation for fuel consumption analysis (ISO11152) for the excavator family of equipments. Our goal is to learn instruction policy model from demonstrations and evaluate the effectiveness of the teaching interface. We obtained six set of demonstrations for

the truck loading task from a human expert. Each demonstration involved filling up of the truck with sand, for which joint positions  $x$ , joint velocities  $v$  and end-effector position  $s^e$ , sampled at 25Hz were recorded. These demonstration trajectories from the excavator model is used to learn the instruction policy in three steps.

In the first step we perform action primitive based segmentation of demonstration trajectories, according to the process described in section III-B. Segmentation of three demonstration set is shown in figure 6. Each of the figure depict two truck loading cycles for clarity although each demonstration consisted of 5 – 6 cycles required to fill the truck to its capacity. We compared our segmentation approach with that of Beta process auto-regressive HMM of [14] in terms of computation time (table I) required to segment on a i7-6700K CPU @4GHz and 24 GB RAM machine. Computation time required by BP-AR-HMM scaled geometrically with data size, in comparison action primitive based segmentation scales almost linearly. Being computationally efficient action primitives can be used to analyze and infer tons of trajectory data from demonstrations. Each action primitive segment  $\mathcal{A}_i$  is a state-action pair  $(s_i, a_i)$  where  $s_i$  is the end-effector pose  $s^e$  from which the action primitive segment began, and  $a_i$  is the associated action primitive. Thus using segmentation procedure we obtain a set of state-action pairs  $\mathcal{O}_j = \{(s_1, a_1), (s_2, a_2), \dots, (s_{N_j}, a_{N_j})\}$ , where  $N_j$  is the total number of action primitive segments observed in  $j^{th}$  demonstration. Each set  $\mathcal{O}_j$  is used as an input to the DPMIRL algorithm, this is the second step which decomposes the task space into finite subgoals. This decomposition is crucial for instruction policy model which consists of action primitive sequence that guides the human operator from one subgoal to another. One such decomposition that results into six subgoals, three each w.r.t the two task objects (pile and truck) is shown in figure 7a. Note that clustering is based on euclidean distance metric defined on a four dimensional vector  $((x, y, z)$  coordinate w.r.t the task object and the bucket angle), which explains the co-location of clusters in the figure, which are actually far apart in the fourth dimension of bucket angle. We followed similar procedure using BNIRL approach [11] for comparison, and the result is shown in figure 7b. Since the state  $s_i^e$  of action primitive segments are not co-located with the subgoal location, decomposition obtained using the latter method is not suited for the instruction policy model. Hence as a third step we utilized the decomposition generated by DPMIRL to learn the parameters of instruction policy model (section III-D) which is then used to guide human operators in performing the truck loading task.

### C. Testing Instruction Policy Model

We tested the efficacy of the proposed instruction policy model by guiding novice operators in the performance of truck loading task in novel configurations. The process of instructing a human operator while performing a task is depicted (in green) in figure 1. At any given time instant  $\tau$ , task space configuration comprising of end-effector position,

dirt pile and truck position, joint positions and base frame position is available to the co-robot. Co-robot then generates the action primitive  $Z_\tau^i$  using the instructional policy model, based on the previous action primitive  $Z_{\tau-1}^i$  (calculated from joint position history) and the most likely current subgoal  $X_i$  (subgoal closest to the current end-effector position), using equations (8)-(10). Figure 4 shows the two different instruction strategies that are used by the robot to communicate the action primitive to the human operator, who then controls the actuator movement through joystick input.

A total of 113 participants volunteered under IRB guidelines for the experiments and were split into three groups in order to test the hypothesized visual interfaces. Participants were randomly sorted into groups: Group 1 using the GUI Circles (4b), Group 2 using the GUI with Speed Bars (figure 4a), and Group 3 with no GUI (control). Each participant was instructed briefly on how the controllers work then given a minimum of three trials to attempt to scoop sand from the tub and deposit into the truck using the controls and the assistance of the selected training GUI. Participants were timed and videoed and their performance was evaluated in terms of cycle time (completion time for each cycle), number of action primitives executed per cycle, number of erroneous action primitives executed per cycle and the dump height. Dump height is the height above the truck at which the sand is dumped, and is a measure of preciseness. Since each participant performed more than three cycles of truck loading task, an average of these quantities is presented in figure 8. These results indicate that the instruction policy statistically significantly improves the performance of novice operators across all the parameters. Using instructions from visual interfaces also helped operators in maintaining a lower dump height which is essential in reducing spillage. Interestingly, these results show no significant difference between the two GUI types utilized, a substantial inference that heavily supports design of simple GUIs that focus on generic rewards rather than complex GUIs that focus on specific instructions.

## V. CONCLUSION

This paper presented a new method for tackling a class of inverse learning problems in co-robotics where the robot must learn from expert demonstration for teaching non-expert humans. The main contribution of the work presented is a generalizable and scalable technique, that enables the robot to directly learn a instructional policy from expert demonstrations. The paper introduced the notion of action primitives for decomposition and representation of multi-input-multi-output trajectories of complex multi degree of freedom robots. The main advantage of using action primitives, instead of motion primitives which are typically parameterized in the trajectory space, is that action primitives are simpler for the robot to explain to the humans. Furthermore, action primitives can be used as building blocks for a variety of similar tasks utilizing the same base actions and can be generalized to robots with different scales but same joint space, such as other excavators with different boom, bucket, and arm lengths. Finally, action primitives

lead to efficient unsupervised clustering of possible robot actions. We demonstrated that utilizing action primitives in a nonparametric unsupervised clustering framework leads to an instructional policy that can utilize current robot pose and subgoal as a feedback signal to provide the correct instruction. Two different interfaces for providing instructional feedback to the human learner were validated in a meticulously constructed large human-robot experiment with 113 human participants. Our results clearly show that there is statistically significant difference in learning rate, task performance times, and retention between guided and unguided operators. Interestingly, our results also show that with the feedback based instructional policy in place, even simpler operator instruction interfaces perform as well as complex interfaces.

## REFERENCES

- [1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.
- [2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
- [3] Jesse Butterfield, Sarah Osentoski, Graylin Jay, and Odest Chadwicke Jenkins. Learning from demonstration using a multi-valued function regressor for time-series data. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 328–333. IEEE, 2010.
- [4] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. Joint modeling of multiple related time series via the beta process. *arXiv preprint arXiv:1111.4226*, 2011.
- [5] Daniel H Grollman and Odest Chadwicke Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 261–266. IEEE, 2010.
- [6] Michael Harré, Terry Bossomaier, and Allan Snyder. The development of human expertise in a complex environment. *Minds and Machines*, 21(3):449–464, 2011.
- [7] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1398–1403. IEEE, 2002.
- [8] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, page 0278364911428653, 2011.
- [9] Brian Kulis and Michael I Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 513–520, 2012.
- [10] Bernard Michini, Mark Cutler, and Jonathan P How. Scalable reward learning from demonstration. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 303–308. IEEE, 2013.
- [11] Bernard Michini, Thomas J Walsh, Ali-Akbar Agha-Mohammadi, and Jonathan P How. Bayesian nonparametric reward learning from demonstration. *IEEE Transactions on Robotics*, 31(2):369–386, 2015.
- [12] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2):79–91, 2004.
- [13] Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. *arXiv preprint arXiv:1206.5264*, 2012.
- [14] Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, 2013.
- [15] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5239–5246. IEEE, 2012.
- [16] T. J. Nokes, C. D. Schunn, and Michelene T. H. Chi. Problem solving and human expertise. In Penelope Peterson, Eva Baker, and Barry McGaw, editors, *International Encyclopedia of Education*, volume 5, pages 265–272. Elsevier, Oxford, 5 edition, 2010.
- [17] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.
- [18] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51(61801):1–4, 2007.
- [19] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.
- [20] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer, 2006.
- [21] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2007.
- [22] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438, 2008.



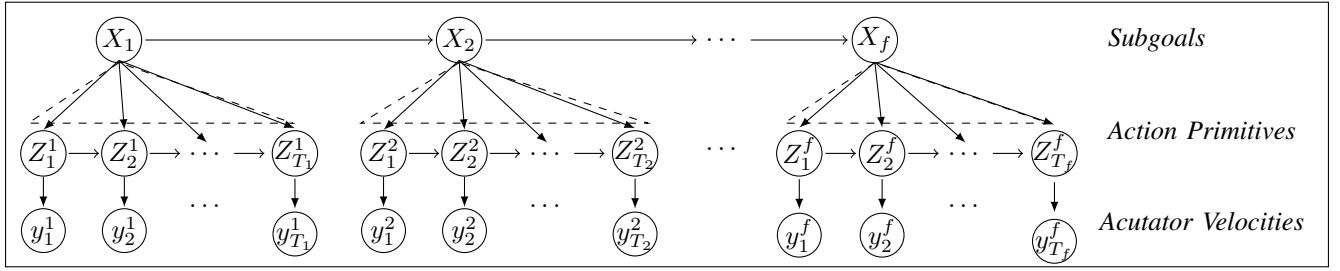


Fig. 3: Task-come-Instruction model obtained based on subgoals generated using DPMIRL and the action primitive segments.

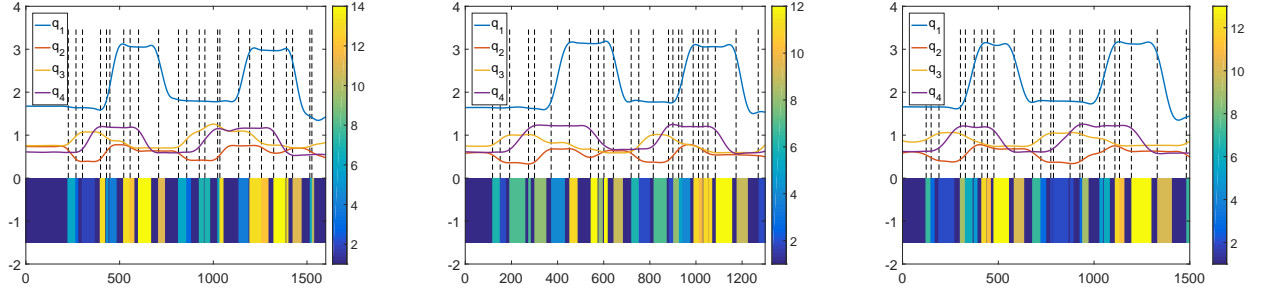


Fig. 6: Segmentations of three demonstration trajectories for two cycles of truck loading task. Each demonstration had five cycles, only two are shown for clarity. Position of each joint  $q_i$  is sampled 25 times per second. Action primitive labels at each time step are indicated by unique colors. Grid lines indicate the starting point of action primitives except for the noisy perturbation action primitive represented in dark blue.

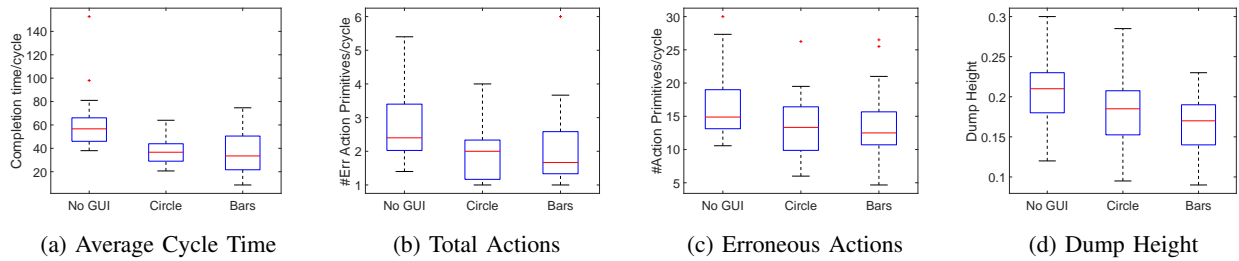


Fig. 8: Instruction policy performance results in terms of the four parameters obtained from testing of 113 participants who randomly distributed into three control groups: i) No GUI, ii) Circle and iii) Bars.